An approach to extract RESTful services from web applications

Bipin Upadhyaya* and Ying Zou

Department of Electrical and Computer Engineering,
Queen's University,
19 Union Street, Kingston, Ontario, Canada
Email: 9bu@queensu.ca
Email: ying.zou@queensu.ca
*Corresponding author

Foutse Khomh

SoftWare Analytics and Technologies Laboratory (SWAT),
Polytechnique Montréal,
Montréal, Canada
Email: foutse.khomh@polymtl.ca

Abstract: The web is the largest database with a huge amount of information and services primarily intended for human users. A user performs different tasks on the web, such as reserving a table in a restaurant. The reuse of web application components would offer greater productivity and ease the maintenance of web applications. The focus of this paper is to circumvent this limitation by proposing an approach to interactively identify reusable web tasks in a web application. We perform a case study on 21 real world web applications from four domains. We identify tasks and services from these web applications. Results show that our proposed approach can identify tasks correctly with a precision of 89% and a recall of more than 90%. Our proposed approach identifies relations among tasks with a precision of 86% and 100% recall. Our approach semi-automatically extracts reusable tasks and represent each task as a RESTful service.

Keywords: web application; service extraction; RESTful service; design tools and techniques.

Reference to this paper should be made as follows: Upadhyaya, B., Zou, Y. and Khomh, F. (2015) 'An approach to extract RESTful services from web applications', *Int. J. Business Process Integration and Management*, Vol. 7, No. 3, pp.213–227.

Biographical notes: Bipin Upadhyaya received his PhD degree in 2014 from Queen's University. He is currently a post-doctoral fellow in the Department of Electrical and Computer Engineering in Software Re-Engineering Lab in Queen's University. His research interest includes service-oriented architecture, user service composition and cloud computing.

Ying Zou is a Canada Research Chair in Software Evolution. She is an Associate Professor in the Department of Electrical and Computer Engineering and cross-appointed to the School of Computing at Queen's University in Canada. She is a Visiting Scientist of IBM Center for Advanced Studies, IBM Canada. Her research interests include software engineering, software reengineering, software reverse engineering, software maintenance and service-oriented architecture.

Foutse Khomh is an Assistant Professor at Polytechnique Montréal, where he heads the SWAT Lab on software analytics and cloud engineering research (http://swat.polymtl.ca/). He received his PhD in Software Engineering from the University of Montreal in 2010. His research interests include software maintenance and evolution, cloud engineering, service-centric software engineering, empirical software engineering, and software analytic. He has served on the program committees of several international conferences including ICSM, WCRE, MSR, ICPC and SCAM, and has reviewed for top international journals such as *SQJ*, *EMSE*, *TSE* and *TOSEM*. He is Programme Chair for Satellite Events at SANER 2015, and Programme Co-Chair for SCAM 2015. He is one of the organisers of the RELENG workshop series (http://releng.polymtl.ca) and Guest Editor for a special issue on release engineering in the *IEEE Software Magazine*.

This paper is a revised and expanded version of a paper entitled 'Extracting RESTful services from web applications' presented at the Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Taipei, Taiwan, 17–19 December 2012.

1 Introduction

Millions of web applications are available and even more people access web applications for conducting their daily activities, such as buying a product or booking a flight ticket. Web applications are popular, owing to the ubiquity of web browsers and the possibility to update and maintain web applications without interrupting the clients. A web application is coded in a browser-supported language, such as JavaScript, and combined with a browser-rendered markup language, such as the hypertext markup language (HTML). Web applications are facing new challenges in today's business environment, such as the integration of software provided by different organisations. In the current state of practice, most web applications are intended for manual use. Because of shorter time-to-market, most web applications are not built using principles of service-oriented architecture (SOA), which results in the web applications lacking machine-to-machine interaction capability. Similar to the problems faced by early traditional applications, many web applications become legacy systems because of their lack of machine-to-machine interaction capability. Moreover, these web applications are full of unstructured data that make them hard to process automatically (Lomotey and Deters, 2014). There is a desperate need for tools that can convert unstructured web data into structured information, by creating an automatically processable machine to machine functionality. We believe this need can be fulfilled through web services which provide machine-to-machine interactions.

A web service is a software component designed to support interoperation between machines over the web. Web services are increasingly used as basic constructs for rapidly developing low-cost distributed applications. The composition of services via business processes are covered by existing tools and solutions, concepts for lightweight service consumption are still in a preliminary phase. The complexity of state-of-the-art SOA technology prevents users with limited IT skills in getting easy access to web services and their offered functionalities. In a service chain, one web service invokes another based on the service definition. For some web services, the invoker may be a user. The ways for a service to interact with an application and a user should be different. When the service interacts with a user, it is preferable to provide a UI. A user may access a web service from different devices, such as desktops and various handheld devices, and the UI should be designed differently to fit different characteristics of the devices. However, it would be a burden to a service composer to manually develop the interfaces for potential user interactions or even manually design various interfaces to fit potential user devices. The two most popular architectural styles used in web services are simple object access protocol (SOAP)-based services and RESTful services. Compared with SOAP-based services, RESTful services are lightweight and easy to build, and provide readable results. Hence, the majority of service providers now uses RESTful services (Programmable Web, http://www.programmableweb.com/apis).

It is not simple to use data and functionalities from different web applications as web services. A little work has been done on the migration of web applications to SOA. The few migration approaches that have been proposed either migrate a web application into SOAP-based service or needs to re-engineer the web application's code-base (Almonaies et al., 2010; Tatsubori and Takashi, 2006; Ajlan and Zedan, 2007; Khadka et al., 2013). However, the web application code base is not always available and SOAP-based services consume more power and time to process the messages (Upadhyaya et al., 2011). In this work, we view a web application as a set of tasks; an activity as a set of task and a process as a set of activities. Contrary to previous work, we do not re-engineer the web application code base, but instead, we reverse engineer the client representation, request and response pattern of a web application, to extract tasks as services. A task is a goal specific functionality, such as search a restaurant, and book a table in a restaurant. A goal may be defined as a state of affairs that a user wishes to achieve; a task is the course of action a user goes through in order to achieve this state. In a web application, the code responsible for a task is scattered between several files, and it is written in different languages, such as service side scripting language (e.g., PHP), database query languages (e.g., SQL), client side scripting language (e.g., JavaScript), and HTML. We treat web applications like black-boxes and extract tasks by analysing client-side representation of a web application. Extracting a task from a client representation can be particularly useful when the code-base of a web application is not available. The extracted tasks can then be specified in terms of RESTful service and deployed through proxies accessing the original web server and parsing its responses.

This paper extends our earlier work published in the proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA) (Upadhyaya et al., 2012). We enhance this previous publication in the following aspects:

1 An improvement of the model used to represent a task. We can now identify a segment of a web application that a user browses to accomplish a task. We represent the functionality of the identified segment of a web application as a RESTful service.

2 The extraction of logical data as input and output for a task, from the data decorated with HTML tags for human users. The extracted data capture semantic information; making machine interactions easier.

3 The identification of relations between tasks to automate the transition between tasks and the migration of tasks as RESTful services.

4 A case study that examines the performance of our proposed approach for extracting service descriptions from web applications. The case study examines the proposed approach on web applications from four domains (i.e., book, housing travel and tourism).

The remainder of this paper is organised as follows. Section 2 discusses background knowledge on web applications and RESTful services. Section 3 introduces a meta-model for tasks. Section 4 presents an overview of our approach. Section 5 reports our case study and discusses its results. Section 6 discusses the related works. Finally, Section 7 concludes this paper and explores some avenues for future work.

2 Background

In this section, we introduce web applications and RESTful services in more details. Our paper focuses on extracting RESTful services from web applications, hence understanding the following two topics is crucial.

2.1 Web applications

In web applications, the layout of a web page is defined using HTML and is represented with the document object model (DOM). All client side interactions involve modifications of the DOM. The presentation of the web pages is defined through cascading style sheets (CSS). Figure 1 shows a code segment from a web application where the style is defined by the CSS class input label, and JavaScript functions. A web application uses the hypertext transfer protocol (HTTP) as the transfer protocol and consists of a series of events. An event is a subset of an application that consists of at least one user input, followed by some processing. For example, an online banking system should maintain a communication session with a specific user during the time the user is logged in. Unfortunately, the communication protocol between a web browser and a web server (HTTP) is stateless, and it does not provide functionality on session control. The connection is established when a browser sends out a request and receives a response message. In order to maintain a logical session between a web browser and a web server, the identification of a web client should be included in each request/response communication cycle. The cookie technique has emerged as a solution to enable user's control of the session. A cookie contains a unique identification information that enables a web server to recognise the identity of the browser and trace the communication with the client.

The conceptual modelling of most existing development methods for web applications are based on the objects (or data) and the related methods, functions, or services. A web page frequently adopts the traditional create, read, update, and delete (CRUD) pattern: a web page is limited to basic operations on objects and their relationships. A web developer manually links different web pages to enable the navigation of users from one web page to another during the accomplishment of their goal.

Figure 1 A fragment of client code segment of a web application

```
<html>
<script>
  function validateForm(){
  var x=document.forms["myForm"]["FName"].value;
  if (x==null || x==""){
   alert("First name must be filled out");
   return false;
   }}
</script>
 ...
 <form name "myForm" onsubmit="return
 validateForm()" action="..">
 <div class="inputlabel">First name</div>
 <input type="text" name="FName" />
 ....
<input type="submit" value="Submit">
.....
</html>
```

2.2 RESTful services

A RESTful service is a web of interconnected resources identified with URIs. A RESTful service can be manipulated through a uniform interface (e.g., HTTP operations), whose state is served through representations (e.g., an HTML page). A resource representation embeds links and controls for a service. The simplicity of REST (Fielding, 2000), along with its natural fit over HTTP, has contributed to its status as a method of choice for exposing the data of web applications. At the core of REST based design is a set of state transfer operations universal to any data storage and retrieval systems. A resource may have two states: a server state and a client state. We are interested in client side state and the changes during the completion of a task.

Hypermedia as the engine of application state (HATEOAS) is a constraint of the REST application architecture. The principle is that a client interacts entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge to interact with particular applications or server beyond a generic understanding of hypermedia. One of the benefits of using RESTful services is the ability to use HTTP Headers to provide the context of request around each of the CRUD operations. A request to a particular resource might result in an HTML, XML, or JSON depending on the desired media type transmitted in the HTTP Accept header. This allows developers to overlay the programmatic API for a website directly on top of the site exposed to web users and reduces the cost and complexity of providing multiple formats for accessing the underlying data of a website. The function signature of a call in RESTful services is described by the tuple: (Resource URL, HTTP method, Input parameters and Accept Header).

The web application description language (WADL) (http:ééwww.w3.org/Submission/wadl/) is a machine-readable XML description of RESTful web services. WADL models the resources provided by a service and the relationships between them. WADL is intended to simplify the reuse of RESTful services. It is platform and language independent and aims to facilitate the reuse and integration of RESTful services.

3 Meta model for a task

A task is a set of resource interactions grouped in a meaningful way to accomplish a goal. The identification of a task is essentially centred on the question: What will a user do with a web application? A task is a course of actions that a user performs on a web application. A task captures a navigation structure of a web application that performs certain functionality. A task is identified on the basis of two main characteristics:

1 should be reusable

2 should perform a functionality.

An example of a task includes searching a product, login into a web page and purchasing a product. For example, by identifying a task to buy a product, service providers can reduce multiple user interactions (e.g., product selection, credit card verification and address confirmation) to one task where a user do not have to go through multiple interactions. The types of resources included in tasks define the nature of a task. We have identified three types of resources used in web applications as shown in Table 1. The classification is based on how a user-agent (e.g., web browser) interacts with a resource. Type I resources have fixed URLs. However, the content changes over time or when a user invokes a URL. An example of Type I resource is a weather page (Weather Forecast, http://www.theweathernetwork.com/weather/caon0349).
Type II resources take input as URL parameters or payload. The representation of a resource updates with the changes in the parameters. An example of Type II resources is a product search page of an ecommerce site (EBay, http://www.ebay.ca/sch/i.html?_nkw=IPhone). Type III resources use both input and client side code to manipulate and change the resource state. A user event, such as a button click, calls a JavaScript function. The HTTP protocol is invoked from the JS function. An example of Type III is a login page that validates the format of a user input before requesting a resource (Google Accounts, https://accounts.google.com/ServiceLogin). A task can be accomplished by one or more resource interactions as shown in equation (1). Examples of resource interaction include opening a page specified by a URL, clicking a link, and submitting some data to a form. Each resource interaction is a function of a URL, http method, input parameters or a client side script as shown in equation (2).

$$Task := RI \mid RI "." Task \tag{1}$$

$$RI := f(URL, method, ip) \mid execute(S) \tag{2}$$

where *RI* is the resource interaction. *ip* is the input parameters, method is a HTTP method used on *URL* and *execute* (*S*) is the execution of a client side code *S*.

Table 1 Different types of resources in a web application

Resource type	*Description*	*Example*
Type 1	Type I resources represent a simple process without any input parameters	Information web pages, such as a web page without any parameters
Type II	Type II resources take input parameters and output representation	Searching products based on keywords, such as Amazon product search web page
Type III	Type III task is a complex process with input parameters, output representation and client side scripts	A login process with client-side validation, such as an e-mail login process

Figure 2 shows the meta-model to model users' tasks. A task starts with an initial resource (i.e., initial state) and ends with one or more final resource. Each resource has a URL, HTTP method, request and response. The request and response contain the header information, input parameters and response representation. A response representation is the description of the messages sent or received from a resource in terms of a technological language. Currently, XML and JSON are the most popular languages for describing these messages. Therefore a representation is defined in the meta-model as an abstract entity that is generalised in the different types of representations according to the corresponding media-types. For accessing or modifying a resource, one of the four HTTP methods (GET, POST, PUT or DELETE) are used. HTTP headers define the operating parameters of a resource interaction. While completing a task, a resource undergoes a series of transitions. A transition can be triggered by a user action (e.g., form submission and resource request) or by system events (e.g., automatic updates of the representation at certain intervals and web page redirections). Resources in our meta-model can be one of the three resources listed in Table 1. Based on the model presented in Figure 2, we describe each task as a RESTful resource.

Figure 2 Meta-model for users' tasks

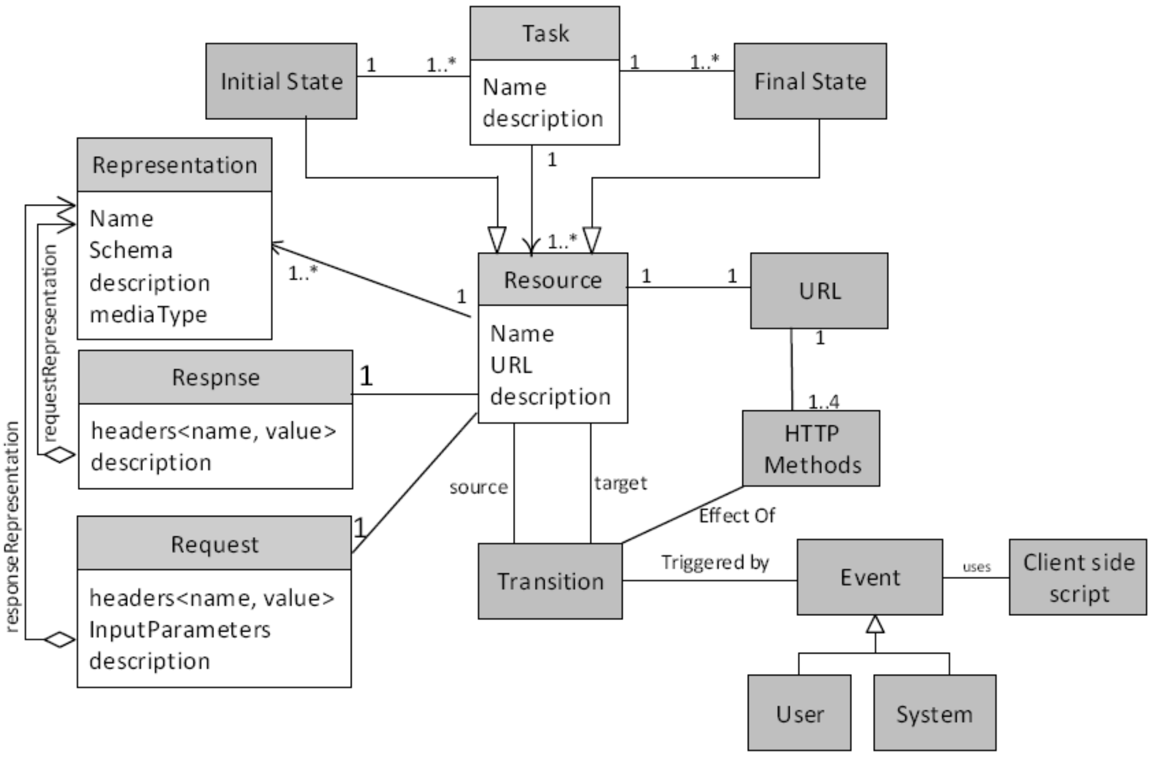
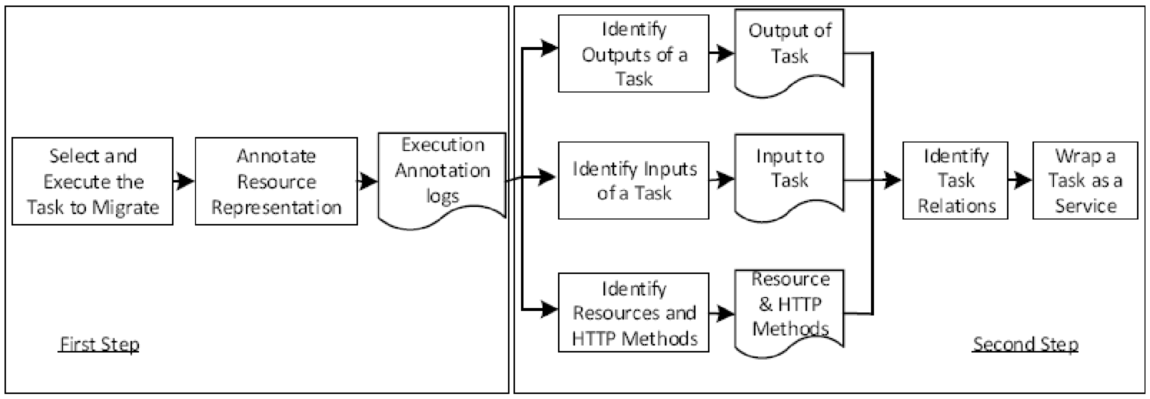


Figure 3 Overview of our approach to identify services from a web page


Select and Execute the Task to Migrate
Annotate Resource Representation
Execution Annotation logs
First Step
Identify Outputs of a Task
Output of Task
Identify Inputs of a Task
Input to Task
Identify Resources and HTTP Methods
Resource & HTTP Methods
Identify Task Relations
Wrap a Task as a Service
Second Step

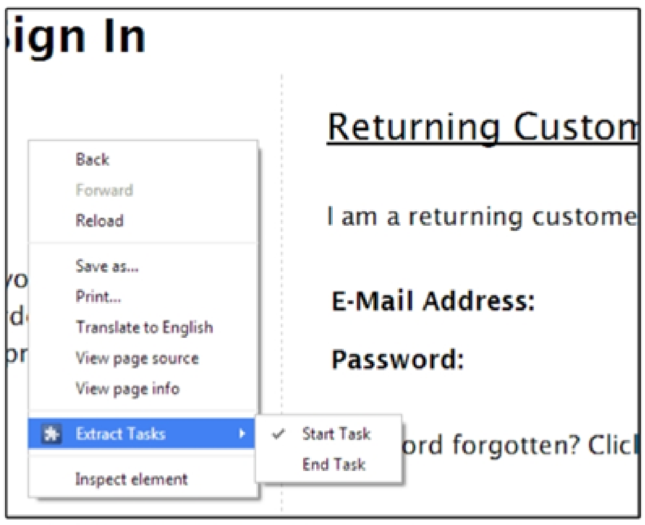
4 Our approach

Figure 3 gives an overview of our approach to represent a task as a RESTful service extracted from a web application. Our approach consists of two steps as shown in Figure 3. The first step is to select and execute a task to migrate. A user does not necessarily need to be an expert in the language and technology used to develop the web application. We implemented as a browser plugin to mark the start of a task and the completion of a task. We instrument a browser to log all the events generated by a web application in a client-side (i.e., execution log) in order to capture all scenarios involved in the completion of a task as shown in Figure 4(a). Figure 4(b) shows an annotation tool to select the region of an HTML page as an output. We store this annotation in as annotation log. Section 4.3 describes in details the annotation process and the annotation logs. Figure 4(c) describes a task completion process for a login task. In the login task, a user clicks the login link [i.e., shown as navigational link in Figure 4(c)] and fills a login form. Based on the data entered, the task can reach one of the two final states (i.e., success or failure). A recorded portion (i.e., between start and end of a task) is performed multiple times with different combinations of input parameters. We separate output based on the similarity of the corresponding DOM structure.

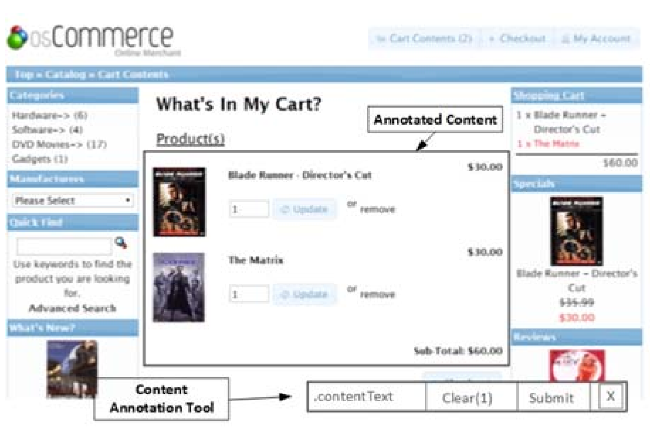
In the case of the example shown in Figure 4(c), the output DOM structure from multiple runs belongs to two groups. One group represents the success, and the other refers to the failure.

The second step is the analysis of the annotation logs and the execution logs to identify input, output and HTTP methods of a task. In the following subsections, we describe the second step (illustrated in Figure 3) in more details.

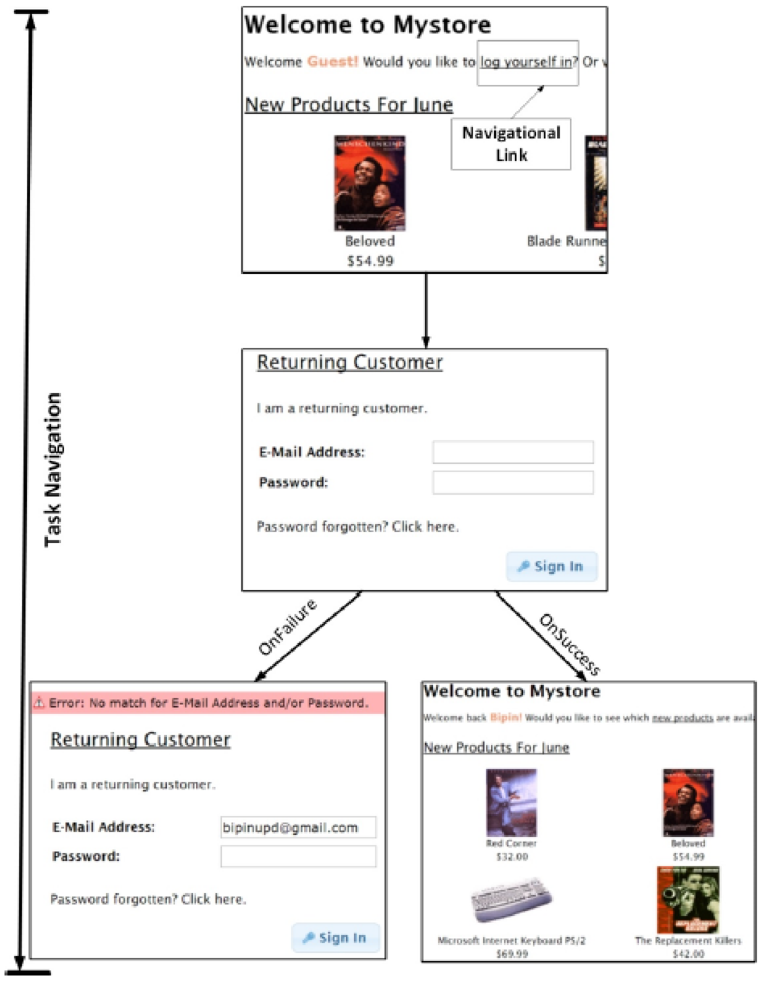
Figure 4 Screenshot showing different phases of task identification, (a) menu to denote start and completion of a task (b) annotated content in a web page (c) task involving navigation over multiple web pages (see online version for colours)



(a)



(b)


Task Navigation
Welcome to Mystore
Welcome Guest! Would you like to log yourself in? Or
New Products For June
Navigational Link
Beloved
$54.99
Blade Runne
Returning Customer
I am a returning customer.
E-Mail Address:
Password:
Password forgotten? Click here.
Sign In
OnFailure
OnSuccess
Error: No match for E-Mail Address and/or Password.
bipinupd@gmail.com
Welcome back Bipin! Would you like to see which new products are avail
Red Corner
Microsoft Internet Keyboard PS/2
The Replacement Killers

(c)

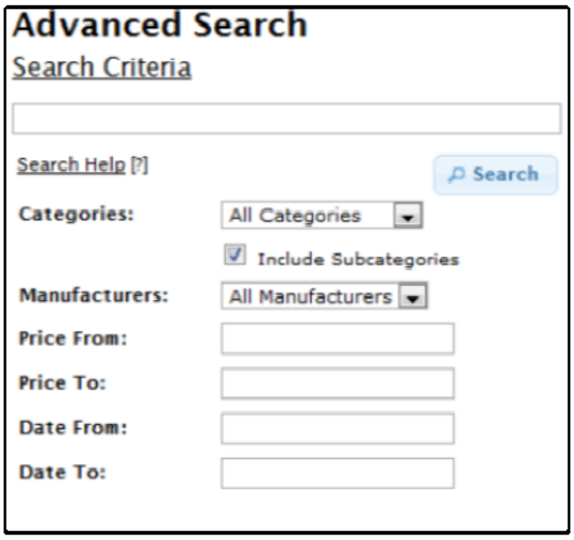
4.1 *Identifying inputs of a task*

A task in a web application includes interactions with web pages. Hence, input parameters for a task include the input parameters needed to accomplish interactions in a web page. These input parameters include input elements in web forms, cookie and request headers. One of the sources of input for a task comes from web forms. Therefore, identifying the correct parameter label is crucial as it corresponds to the data required by a task. In this subsection, we discuss our approach to extract parameters, labels from web forms and other input parameters, such as session and cookie information.

A web form contains different input elements, such as input fields and radio buttons. Each input element contains semantic information (i.e., label) and the name of the element. A label of an input element defines the semantics of that input-element. Hence, correctly identifying a label for an input element helps to retrieve and integrate information hidden behind web form interfaces. As illustrated by meta-model shown in Figure 2, a resource transition can be either a user event or a system event. Web forms and hyperlinks are the most usual ways to provide input to a web application. A web form submission does not always invoke a resource. Web forms generate a number of events. These events are handled by client side functions. Client side functions are executed by a client's web browser and have access, via a DOM, to the resources of the browser, in particular, to the HTML document shown in the browser. For this purpose, the document is represented as a hierarchical object structure where the attributes of each object can be accessed or manipulated by the standard 'dot notation'. For instance, the class identifier (whose meaning is usually defined in a style sheet) of an object element in an HTML document can be changed to myStyle by the assignment *elem.className* = 'myStyle'. JavaScript programs are usually executed by the web browser when some events occur. For instance, if an input button in an HTML form has an attribute *onsubmit* = "*fun*(*x*)", the function call *fun*(*x*) is evaluated whenever the user clicks this button. Our plugin tracks all the events generated during the completion of a task including JavaScript events.

Web forms and hyperlinks contain semantic information (i.e., labels). It is challenging to identify labels that describe HTML input elements. Especially, web forms have different layouts. The positions of labels in a web form depend on the designer of the web form. Labels can be placed above, below, to the left, or to the right of an input element. To identify the label representing an input element, we analyse the content of a web page delimited by the opening and the closing tags of an HTML partitioning element that separates the different sections of a web page. For example, paragraph tag (i.e., <p>) separates a paragraph in HTML. The text nodes under the partitioning element are part of the same blob (i.e., a text contained within a partitioning element). However, style tags, such as, the italic tag (i.e., <i>) and the bold tag (i.e., <b>) add styles (e.g., bold, and italic) within a section of text. Therefore, styling tags are not considered as partitioning elements. A web form is a hierarchy of HTML tags. Labels and form input elements are often positioned in proximity in the hierarchical structure of tags. Hierarchically nested labels and form input elements are placed close to the lowest common ancestor in the hierarchy. If a label and the associated input element are in the same parent structure, they are close to each other within the parent structure. The hierarchical proximity between the elements helps to associate the input elements with the text blob. Figure 5(a) shows a screenshot of a web query interface. Figure 5(b) shows a fragment of the DOM tree of the query web form shown in Figure 5(a). In Figure 5(b), the input field r1 is in closer hierarchical proximity with the label l1 (i.e., 'search criteria') than the label l2 (i.e., 'categories'). Therefore, the label l1 (i.e., 'search criteria') should be associated with the input r1.

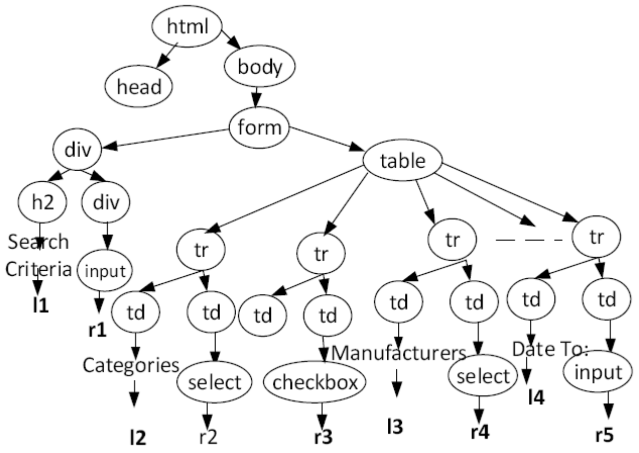
Figure 5 HTML and DOM representation of a web interface, (a) HTML representation of a web form (b) DOM tree of the web form shown in (a) (see online version for colours)


Advanced Search
Search Criteria

Search Help [?]
Search

Categories:	All Categories
	✓ Include Subcategories
Manufacturers:	All Manufacturers
Price From:	
Price To:	
Date From:	
Date To:	

(a)



(b)

To identify the association between input elements and labels, we traverse and analyse the DOM tree to find the text nodes that constitute a label. When a partitioning element (e.g., paragraph tag <p>) is reached, we create a new label. The text node under the partitioning element is added to the label. If the partitioning element contains another partitioning element as a child, then the text nodes that appear under the sub-partitioning child belongs to the text blob of the sub-partitioning child. For each input element, we compare the hierarchical proximity between the input element and the text blob. For example, in Figure 5(b) to reach r1 from l1, we have to traverse three nodes (i.e., h2, div and div). The label with the least distance is considered as a candidate of an input description tag for the input element. The distance between a text blob and an input element is given by the number of nodes visited from the text blob to reach the input element. For example, in Figure 5(b), the distance between the nodes r1 and l1 is 3; the distance between the nodes, r1 and l2, is 6; and the distance between the nodes, r1 and l6, is 6. The node r1 has the least distance with the text blob l1, and hence, the node l1 is selected as the description tag for the node r1. If more than one candidate is identified, we calculate the edit distance (Baeza-Yates, 1989) between the candidates and the 'name' attribute of the input element to choose the candidate for the input description tag. The edit distance between two strings of characters is the number of operations required to transform one string of characters into the other.

Cookies are considered as an input field in HTTP requests. The web browser cookie technology provides persistent data storage on the client side. A cookie is a dataset consisting of at least a cookie's name, a value and a domain. Cookies are sent by a web server as part of an HTTP response message using the set-cookie header field. The cookie's domain value is used to determine in which HTTP requests the cookie is included. Whenever the web browser accesses a webpage that lies in the domain of the cookie, the cookie is automatically included in the HTTP request. Cookies are often used as authentication tokens in web applications. After a successful login procedure, the server sends a cookie to the client. Every following HTTP request that contains this cookie is automatically regarded as authenticated. We track changes in cookies and other HTTP header fields and consider them as input parameters.

4.2 *Identifying outputs of a task*

Multiple web pages constitute an output of a task. As a user finishes a task, he annotates the output from web pages. In this subsection, we describe the process of data extraction from web pages for a task.

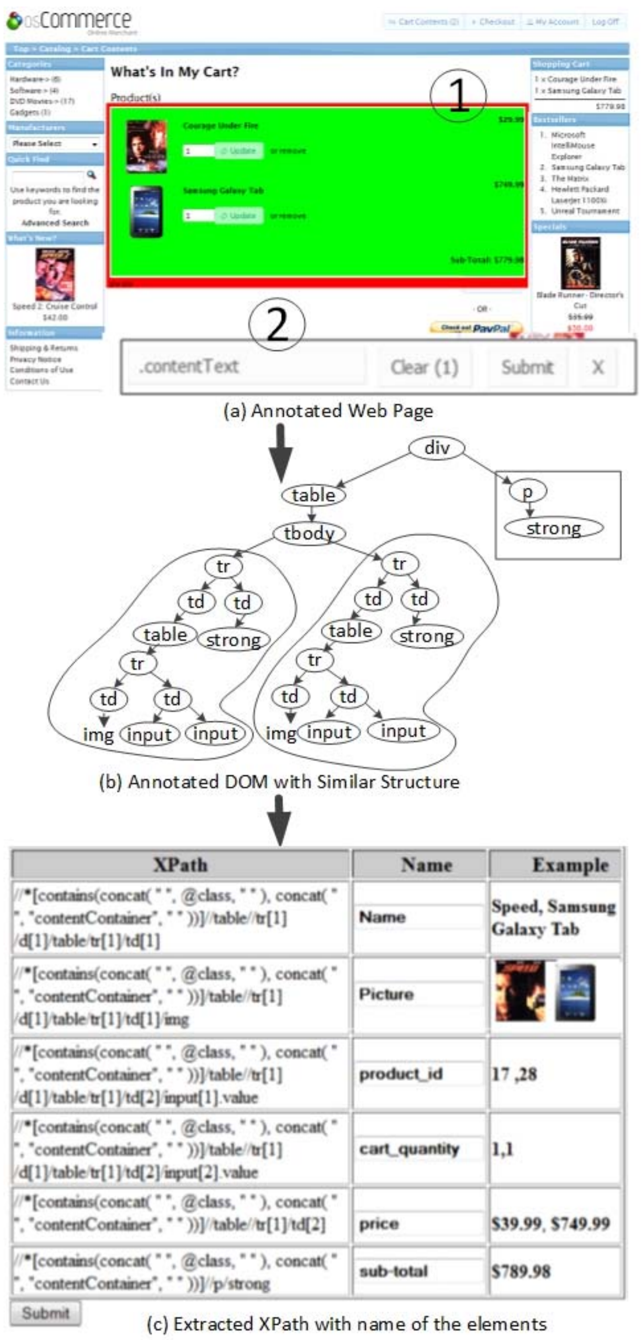
The output of a task is encoded in the return representation generated once a web page is requested or a web form is submitted. There are mainly two kinds of data that are received in a return page:

1 header information containing a status code

2 a resource representation.

We record the changes in the header fields. To identify the output of a task, a user has to select the region in an HTML representation that represents the output of the task, using our plugin. In a resource representation, we look for data relevant to a task output. For example, in Figure 6(a), the items in the shopping cart relevant are the name of the product, quantity and the unit price. The return page is represented in a common template. The generated template content contains information, such as, advertisements, and navigational panels.

Figure 6 Identify data segments from an HTML representation (see online version for colours)



XPath	Name	Example
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[1]/table/tr[1]/td[1]	Name	Speed, Samsung Galaxy Tab
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[1]/table/tr[1]/td[1]/img	Picture	
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[1]/table/tr[1]/td[2]/input[1].value	product_id	17 ,28
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[1]/table/tr[1]/td[2]/input[2].value	cart_quantity	1,1
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[2]	price	$39.99, $749.99
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//p/strong	sub-total	$789.98

Although the above mentioned parts of a web page may be helpful for a user to browse, they can be considered as 'noisy data' that may complicate the process of extracting the output of a task from web pages. The noisy data could be wrongly matched as correct data resulting in either inefficient or even incorrect wrappers. Therefore, we allow users to identify the data rich section that contains the output of a task. Our data extraction is based on the following steps:

1 Select a portion of an HTML representation that contains the output of a task. Figure 6(a) shows an example of a user selecting a specific part of an HTML page.

2 Parse the HTML document to find the starting (SP), and ending (EP) positions of the selected region.

3 Identify regions with the similar DOM structures between SP, and EP. Our approach identifies segments of DOM regions with the similar DOM structure. The similar DOM structure represents the similar type of data. Figure 6(b) shows an example of similar DOM structures. We use the following heuristic to identify the semantics of the extracted elements:

   a Match if web form labels are presented in the response representation.

   b Search for labels in table headers in a resource representation. HTML specifications define tags, such as, header cells in HTML tables, <TH>, and header contents in HTML tables, <THEAD>. We list the columns of HTML tables.

   c Search for voluntary labels encoded in the response pages. For example, if a response page contains a column with the symbol '$', we consider that the data item represents currency related fields such as, price.

4 Refine the semantics of the extracted data template. A user can import the available ontology or define her own ontology if there is no available ontology. Figure 6(c) shows a screenshot of GUI that helps a user to refine the extracted data template.

Figure 6(c) shows a screenshot of the GUI of our tool which is used to help users to refine the extracted data templates. Based on user selections, we identify different parts of the resource representation. The selected part of a resource representation is extracted as an XPath. The semantics for each XPath is defined using a label extracted from the web page. A user can verify each semantic label, and XPath mapping as shown in Figure 6(c). The XPath, and semantic label mapping are described in a single file for each task.

4.3 Identifying resources and HTTP methods of a task

At this stage, we identify resources required to accomplish a task, and the execution sequence between the resources. A task often encapsulates multiple resources. When abstracting different resources to a task, the focus is the idempotent HTTP methods. Idempotent describes the ability of a method that produces the same result if it is applied to itself (i.e., f(x) = f(f(x)). In HTTP, this concept ensures safely resent of the same representation irrespective of receipt of the same message multiple times. We select unsafe methods over safe methods, and un-idempotent methods over idempotent methods (Fielding, 2000). For example, if a task uses two resources: one using HTTP method retrieves a representation of a resource (e.g., using GET method on a resource URL), and other changes the representation of a resource (e.g., using POST methods on a resource URL), the HTTP method for that task is POST. Similarly, a request/response header of a task is the most recent header used in the resources invoked for that task. The meaning of the HTTP header plays important role in determining the header of a task. For example, *If-Modified-Since* header gives the timestamp when a resource has been changed. Hence, a resource of a task involving multiple resources takes the most recent date among all the dates used in a resource. Similarly, if an intermediate resource changes parameter of cookies during the completion of a task, the most recent change in the cookie is propagated to the client.

4.4 Identifying task relations

We model a task as REST resource; hence we include HATEOS behaviour in extracted services. The HATEOS behaviour will guide a user between different tasks. For example, after register task, a user will be provided with the link for register task. Web developers embed the links that guides a user from one state to another.

We use two different approaches to extract HATEOS information between extracted tasks. The first approach is based on how a user navigates web pages during the task start to completion period. For example, after finishing a *register* task a user performs a login task. Hence, the representation of a *register* resource should contain *login* resources. In addition to that, we analyse all the extracted links and forms in HTML representation that helps to identify the possible next state information. Based on URLs and analysis of representation, we define rules to extract resource. For example, after finishing a *register* task, a user can perform a *login* task. Hence, the *register* representation should contain links for *login* task. In a web application, relations between tasks are embedded as HTML links and web forms. We analyse all the web forms, incoming and outgoing links between the tasks to identify task relations. We propose the following rules to extract task relations from a client-side representation.

4.4.1 Rule 1: identify state changes without requests and responses

This rule identifies the client side script that does not perform HTTP requests, and responses, but change the state for an HTML representation. In such a case, the URL, HTTP-methods, and parameters between the two resource interactions remain the same, whereas there is a change in the DOM elements. This change is due to client side scripts, such as validation of data entered in a web form. For example, in Figure 4(c), when a user tries to submit a form without username, and password, the representation displays an error, but the validation is only performed in the client-side. Hence, the state of the resource representation is dependent on the client-side script.

$$Representation_a = \{URL_a, HTTP-Method, Param_a\}$$
$$Representation_b = \{URL_a, HTTP-Method, Param_a\}$$
$$Representation_a \neq Representation_b$$

The URL, HTTP-method and parameters remain the same, where there is a change in the representation. This kind of change is due to client side scripts. For example, client validation for web forms such as user registration and login forms.

4.4.2 Rule 2: identify tasks sharing output parameters

This rule helps to identify resources that share input output parameters. For example, with a product id, one can find the product as well as the product review as well. The resources may have one to many relationships with other resources. We cluster URLs with similar parameters, and resource paths. To automatically create the clusters, we use k-means (Forgy, 1965; Hartigan and Wong, 1979) which is an unsupervised clustering algorithm. K-means algorithm divides the resources into a set of disjoint groups. The main challenge in using such a clustering algorithm is to identify the expected number of clusters (Hartigan and Wong, 1979). In case of k-means, this parameter is called k. One possible solution is to ask domain experts to identify the proper value for k empirically. However, since we need to automate the process completely, we use a clustering validation approach proposed by Hartigan and Wong (1979). Using this approach, we can measure the success of any possible value for k in generating a set of coherent clusters. To find the proper value for k automatically, we create clusters with all possible values for k where the maximum value is the number of distinct data points. Then, we measure the success of each experiment using Rousseeuw's (1987) approach. Finally, we select the k value with the highest measured success rate for our actual clustering. The resources in a same cluster share input output parameters and hence have resource relations.

$$Resource_a = \{URL_a, HTTP-Method, Parameter_a\}$$
$$Resource_b = \{URL_b, HTTP-Method, Parameter_b\}$$

where

$$Parameter_a = Parameter_b \cup \{x\}$$

Resource$_a$ is related to *Resource$_b$*. For example, if the URL of a product resource (i.e., product Info task) is http://foo.org/product?pid=xx, and the URL of the review resource (i.e., product review task) is http://foo.org/reveiw?pid=xx, the parameter names in the URLs of the product info task, and the product review task are similar, and belong to the same cluster. Hence, the two tasks are related.

4.4.3 Rule 3: identify the next task to perform

A web developer embeds a link or a web form that helps a user to decide what to do next. In this rule, we identify the next task a user can perform after completing a task. We extract all next-state elements. For any two given resource, we choose non-reoccurring elements. A non-reoccurring element is a symmetric difference among a set of next-states. We identify tasks whose initial states are present in the non-reoccurring elements list.

$$Next_a = ExtractFormsLinks\{Task_a\}$$
$$Next_b = ExtractFormsLinks\{Task_b\}$$
$$Next_{common} = Next_a \cap Next_b$$
$$Next_{TA} = Next_b - Next_{common}$$
$$Next_{TB} = Next_a \cap Next_{common}$$

Next$_a$ and *Next$_b$* are the set of links and forms extracted from the representation of Task A and Task B. For example, when a user selects a product and adds to a shop cart, general links (such as categories product links, search forms) get excluded as shown in *Next$_{common}$* above. *Next$_{TA}$* and *Next$_{TB}$* are next state tasks from task A and task B.

4.4.4 Rule 4: identify dependent task through flow sequence analysis

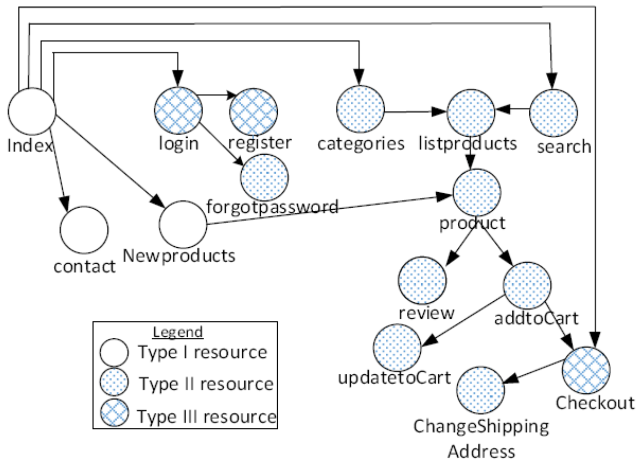
In this rule, we mainly focus in the order of task execution. For two tasks (i.e., task A, and task B), if the completion of task A is required before starting task B, then task B is dependent on task A. One task can be invoked more than once (e.g., multiple add to cart task in a shopping activity). The multiple occurrences of a task should not affect the technique since we do not consider the number of occurrences of a task. If the execution flow of a task is present in another task, a flow-based relation is discovered.

$$Resource_a = \{URL_a, HTTP-Method, Parameter_a\}$$
$$Resource_b = \{URL_a, HTTP-Method, Parameter_b\}$$
$$Resource_a redirects\ to\ Resource_b$$

For example, checking out a shopping cart resource needs a login task to be invoked first; hence checkout task is dependent on the login task.

Different rules may be applied to recognise a common relation. We select unique task relations. Figure 7 shows the resource relations identified among different tasks in a web application. For simplicity, we removed the reverse relation in Figure 7. Figure 7 shows some hierarchical relations as well. In hierarchical relation the parent task has to be complete to perform child task. The register node cannot be reached directly from the index node. *Login* node points to the *register* resource, and *forgotpassword* resource. Similarly, one can only reach a review state from the product or a *Newproducts* resource.

Figure 7 Task relations extracted from a web application in e-commerce domain (see online version for colours)



We have extracted a task and task relations. Figure 8 shows a visual representation of a product search task. A task is denoted by a URL, HTTP method, HTTP headers, input parameters, output parameters, and task relation links. For simplicity in Figure 8, we show only the task name, input parameters, output parameters, and tasks relations.

5 Case study

In this section, we discuss our case study to evaluate the effectiveness of our proposed approach to extract tasks, and task relations from a web application. For this case study, we selected web applications that already have exposed web services. Selecting the web application with exposed services make task selection easier for our case study. We consider each resource as a task and compare the effectiveness of our approach. More specifically, we aim at accessing:

1 the effectiveness of our approach to extract task from a web application with correct input and output parameters

2 the effectiveness of our approach to identify task relations.

In the following subsection, we discuss the case study in more detail.

5.1 Data collection

We randomly choose 21 web applications from five different domains (i.e., finance, weather, ecommerce, book and travel). We make sure there are services for these web applications. Having services allow us to extract tasks as those described in the service description documents. Table 2 lists the domains and the number of web sites selected to assess our approach

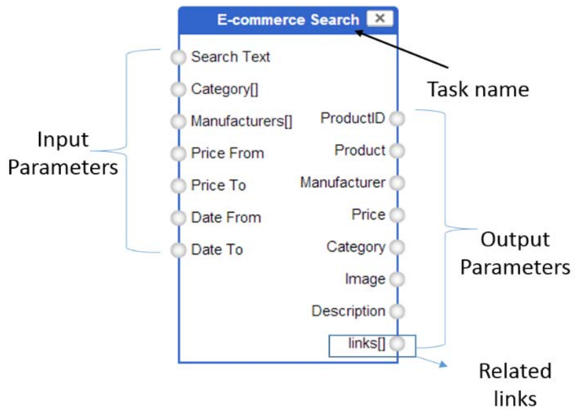
We use our Firefox plugin tool as discussed in Section 4 to extract tasks from web applications and represent the extracted tasks as RESTful services. The plugin tool is designed to assist users in identifying tasks. Figure 3(a) shows an annotated screenshot of the initial and final states of a task. Figure 3(b) helps a user to annotate a resource representation. Figure 6 shows a screen-shot where a user selects a region of an HTML representation and verifies the name of the output elements.

Table 2 Domain and number of web applications used in the case study

Domain	*#*	*Example web application*
Finance	4	http://www.exchangerate.com
Weather	3	http://www.theweathernetwork.com
Ecommerce	6	http://ebay.com
Book	5	http://www.freebooksearch.net/
Travel	3	http://www.expedia.ca/Flights

A user can verify all the identified resources as shown in Figure 9 and generate a WADL file as shown in Figure 10. WADL describes the task extract long with input and output parameters. WADL helps machine to machine integration. The tool is directly connected with the Firefox JavaScript engine. Code fragments specific to some browsers (e.g., Internet Explorer, Opera, and Safari) are not extracted. We execute web applications to record the execution logs and the annotated information. The plugin tool extracts tasks as RESTful services and identifies the relation among different tasks.

Figure 8 A task showing input parameters, output parameters and related links (see online version for colours)



5.2 Evaluation criteria

We measure the quality of our approach using precision and recall. We discuss the evaluation criteria in this sub-section.

5.2.1 Evaluate the quality of tasks extracted from a web application

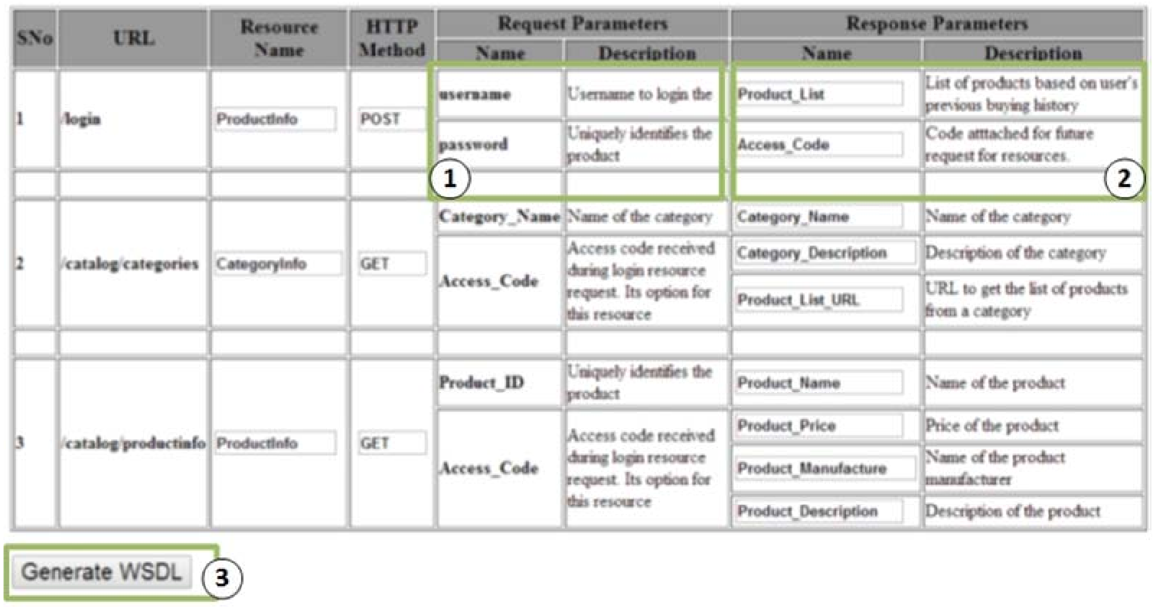
We evaluate the effectiveness of our approach to identify and extract tasks without any prior knowledge of a web application by examining if our approach can correctly identify the input and the output parameters for a task. As defined in equation (3), the precision evaluates if any irrelevant input and output of tasks are misidentified. Recall is a measure of completeness. As specified in equation (4), the recall evaluates whether our approach can correctly identify all necessary input and output of tasks without omissions. We use both metrics to evaluate our approach.

$$precision = \frac{|\{R_i\} \cap \{R_j\}|}{|\{R_i\}|} \tag{3}$$

$$recall = \frac{|\{R_i\} \cap \{R_j\}|}{|\{R_j\}|} \tag{4}$$

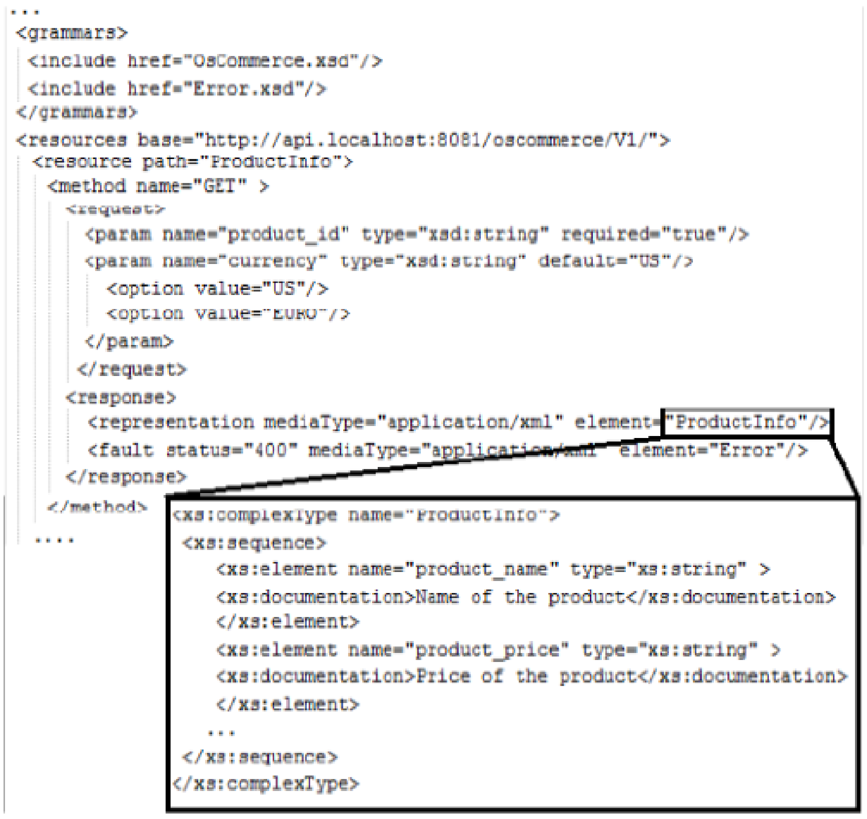
where $\{R_i\}$ the number of input and output parameters identified by our approach; $\{R_j\}$ is the total number of input and output parameters and $\{R_i\} \cap \{R_j\}$ is the number of correctly identified input and output parameters.

Figure 9 An annotated GUI used to verify the tasks identified (see online version for colours)



SNo	URL	Resource Name	HTTP Method	Request Parameters		Response Parameters	
				Name	Description	Name	Description
1	/login	ProductInfo	POST	username	Username to login the	Product_List	List of products based on user's previous buying history
				password	Uniquely identifies the product	Access_Code	Code attached for future request for resources.
2	/catalog/categories	CategoryInfo	GET	Category_Name	Name of the category	Category_Name	Name of the category
				Access_Code	Access code received during login resource request. Its option for this resource	Category_Description	Description of the category
						Product_List_URL	URL to get the list of products from a category
3	/catalog/productinfo	ProductInfo	GET	Product_ID	Uniquely identifies the product	Product_Name	Name of the product
				Access_Code	Access code received during login resource request. Its option for this resource	Product_Price	Price of the product
						Product_Manufacture	Name of the product manufacturer
						Product_Description	Description of the product

Figure 10 A WADL file generated by clicking 'Generate WADL' in Figure 8



```
...
<grammars>
 <include href="OsCommerce.xsd"/>
 <include href="Error.xsd"/>
</grammars>
<resources base="http://api.localhost:8081/oscommerce/V1/">
  <resource path="ProductInfo">
   <method name="GET" >
    <request>
     <param name="product_id" type="xsd:string" required="true"/>
     <param name="currency" type="xsd:string" default="US"/>
       <option value="US"/>
       <option value="EURO"/>
     </param>
    </request>
   <response>
     <representation mediaType="application/xml" element="ProductInfo"/>
     <fault status="400" mediaType="application/xml" element="Error"/>
   </response>
  </method>
  ....
```

```
<xs:complexType name="ProductInfo">
 <xs:sequence>
    <xs:element name="product_name" type="xs:string" >
    <xs:documentation>Name of the product</xs:documentation>
    </xs:element>
    <xs:element name="product_price" type="xs:string" >
    <xs:documentation>Price of the product</xs:documentation>
    </xs:element>
   ...
 </xs:sequence>
</xs:complexType>
```

5.2.2 Evaluate the quality of identified task relations

We evaluate the effectiveness of our approach to extract task relations among identified tasks from a web application. As defined in equation (3) and equation (4), the precision evaluates if there is any irrelevant task relations, while the recall evaluates whether our approach can correctly identify all necessary task relations. As in equations (3) and (4), $\{R_i\}$ is the number of task relations that are correctly identified by our approach; $\{R_j\}$ is the total number of task relations that are manually verified as correct; and $\{R_i\} \cap \{R_j\}$ is the number of correctly identified task relations in a web application. We exercise each web application individually and create a set of tasks containing all the tasks that can be achieved using the web application. The set also contains relations between the tasks [i.e., correct sets from equations (3) and (4)]. We manually verified the correctness of all tasks and task relations that were generated during the data collection phase.

5.3 Analysis of results

In this subsection, we present and discuss the result of the case studies.

5.3.1 Evaluate the quality of tasks extracted from a web application

To identify if our approach can correctly extract tasks, we annotated different tasks in each of the web application shown in Table 2. Table 3 lists the average number of tasks extracted by our approach. We find the extracted resources containing all the three kinds of resources (i.e., Type 1, Type 2 and Type 3). Table 3 lists the results showing the effectiveness of the proposed approach at identifying the input and output parameters. Our approach achieves a satisfactory performance (i.e., above 80% precision) on identifying the input/output of a task. The high recall (i.e., above 95%) shows that our approach can recover most of the input/output parameters.

Table 3 Result of identifying input/output for tasks

Domain	*#Average task identified*	*#Average input and output*	*Precision*	*Recall*
Finance	10	24	80%	93%
Weather	3	6	100%	100%
Ecommerce	12	58	78%	95%
Book	8	74	81%	92%
Travel	12	82	79%	96%

Our approach misidentified some of the input and output element labels because of the complex layout and nested structures of web forms and the response pages. Some of the input elements have default values without having any descriptive text. Similarly, the use of graphic images instead of text hinders the identification of the description of an element.

Table 4 Result of identifying task relations

Domain	*#Average task relations in a domain*	*Precision*	*Recall*
Finance	7	85%	100%
Weather	2	100%	100%
Ecommerce	10	78%	100%
Book	8	77%	100%
Travel	7	82%	100%

5.3.2 Evaluate the quality of identified task relations

Figure 6 shows the task relations identified among different tasks in a web application from the e-commerce domain. Table 4 lists the result showing the effectiveness of our approach in identifying task relations. Indeed, our approach has a satisfactory performance (i.e., above 80% precision) on identifying resource relations. The high recall (i.e., 100%) shows that our approach can recover all resource relations.

Using our approach, a developer can extract services from web applications that have not exposed web services, and also from web applications whose source code is not available.

5.4 Threats for validity

The main threat that could affect the generalisation of the presented results relates to the number of web applications analysed. We have analysed 21 web applications from different domains. Nevertheless, further validation of our approach requires an analysis of a larger set of web applications. We are aware that our approach depends on the tasks selected by a developer. It might be the case that all the tasks have not been selected. Therefore our approach can miss tasks and task relations.

6 Related work

In this section, we discuss migration approaches in web application and web application analysis approaches.

SOA migration is an architectural migration from any non-SOA system to a system that follows the SOA principles, in order to achieve a new maintainable SOA implementation of the system. The major benefits of adopting SOA as a design framework is the ability to realise rapid and low-cost system development, to improve overall system quality, and to better enable integration with other systems. Several studies in the literature have focused on the problem of migrating traditional legacy systems to web services. Lewis et al. (2006) discuss a migration technique called the service-oriented migration and reuse technique (SMART). The SMART technique helps organisations analyse legacy systems to decide whether their functionalities can be reasonably exposed as services in a SOA. In our approach, a user can use a framework like SMART to identify the tasks to migrate.

Sneed and Sneed (2003) discuss a tool-supported method for legacy code written in COBOL and wrapped behind an XML shell allowing individual functions within the programs to be offered as web services to any external user. Tatsubori and Takashi (2006) present a framework named H2W, which can be used to construct web service wrappers for existing, multi-paged web applications. H2W's contribution is mainly in its service extraction step. The authors propose a page-transition-based decomposition model and a page access abstraction model with context propagation. Similar to Tatsubori and Takashi (2006) approach, our approach extracts services analysing the client representation. However, our approach represents a task as RESTful services with the state transition. Almonaies et al. (2011) present an approach to migrate web applications to a web service. Unlike our approach, Almonaies et al. (2011) use analysing the server side code to extract service.

Our work understands a web application by analysing client side code. There are two tools that facilitate the understanding of dynamic web page behaviours: Script InSight (Li and Wohlstadter, 2009) and FireCrystal (Oney and Myers, 2009). Script InSight helps to relate the elements in the browser with the lower-level JS syntax. It uses the information gathered during the script's execution to build a dynamic, context-sensitive, control flow model that summarises tracing information. FireCrystal facilitates the understanding of interactive behaviours in dynamic web pages by recording interactions and logging information about DOM changes, user input events, and JS executions. After the recording phase, a user can use an execution timeline to see the code that is of interest for the particular behaviour. Compared to our approach, they make no attempt to track data dependencies between different resources and are limited to understanding the code. However, we use this information to extract tasks, identify relations among tasks and then wrap tasks as services.

7 Conclusions

In this paper, we present a meta-model to represent tasks as RESTful services. We propose a semi-automatic approach to extract RESTful services from web applications. Our approach migrates reusable tasks extracted from web applications towards RESTful services. We analyse client side web user interfaces and HTML representation developed with a combination of JavaScript, HTML and CSS code. Our approach only requires client-side code of a web applications and do not depend on the server side code. We identify required resources for the tasks from web applications. We find that our approach can identify input/output parameters related to tasks with high precision and recall. More specifically, our approach has 89% of precision and 90% of recall when identifying input/output parameters for tasks. Similarly, 86% of precision and 100% of recall are achieved when extracting task relations. Our work can help to integrate different functionality from web applications that have not been exposed as web services.

In the future, we plan to extend our approach to support the extraction of tasks and their relations from web applications that use Silverlight or Flash.

References

Ajlan, A. and Zedan, H. (2007) 'E-learning (MOODLE) based on service oriented architecture', in the *EADTU's 20th Anniversary Conference*, Lisbon, Portugal, pp.62–700.

Almonaies, A., Alalfi, M., Cordy, J.R. and Dean, T.R. (2011) 'Towards a framework for migrating web applications to web services', *21st IBM Centre for Advanced Studies International Conference on Computer Science and Software Engineering*, Toronto.

Almonaies, A., Cordy, J.R. and Dean, T.R. (2010) 'Legacy system evolution towards service-oriented architecture', in *International Workshop on SOA Migration and Evolution*, pp.53–62.

Baeza-Yates, R. (1989) 'Algorithms for string matching: a survey', *ACM SIGIR Forum*, Vol. 23, Nos. 3–4, pp.34–58.

delicious/help/api – Delicious.com – Discover Yourself! [online] http:///www.delicious.com/help/api (accessed 5 November 2014).

EBay [online] http://www.ebay.ca/sch/i.html?_nkw=IPhone (accessed 15 July 2015).

Fielding, R.T. (2000) *Architectural Styles and The Design of Network-based Software Architectures*, PhD thesis, University of California, Irvine.

Forgy, E.W. (1965) 'Cluster analysis of multivariate data: efficiency vs. interpretability of classifications', *Biometrics*, Vol. 21, No. 3, pp.768–769.

Google Accounts [online] https://accounts.google.com/ServiceLogin (accessed 5 November 2014).

Hartigan, J.A. and Wong, M.A. (1979) 'A K-means clustering algorithm', *Applied Statistics*, Vol. 28, No. 1, pp.100–108.

Khadka, R., Saeidi, A.M., Jansen, R.L. and Hage, J. (2013) 'A structured legacy to SOA migration process and its evaluation in practice', in *IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA*.

Lewis, G., Morris, E. and Smith, D. (2006) 'Analyzing the reuse potential of migrating legacy components to a service-oriented architecture', in *Proc. of the Conference on Software Maintenance and Reengineering*, pp.15–23.

Li, P. and Wohlstadter, E. (2009) 'Script InSight: using models to explore JavaScript code from the browser view', *Proceedings of the 9th International Conference on web Engineering*, pp.260–274

Lomotey, R.K. and Deters, R. (2014) 'Analytics-as-a-service framework for terms association mining in unstructured data', *International Journal of Business Process Integration and Management (IJBPIM)*, Vol. 7, No. 1, pp.49–61.

Oney, S. and Myers, B. (2009) 'FireCrystal: understanding interactive behaviors in dynamic web pages', *IEEE Symposium on Visual Languages and Human-Centric Computing*.

Paulson, L.D. (2005) 'Building rich web applications with Ajax', *Computer*, October, Vol. 38, No. 10, pp.14–17.

Programmable Web [online] http://www.programmableweb.com/apis (accessed 5 November 2014).

Rousseeuw, P.J. (1987) 'Silhouettes: a graphical aid to the interpretation and validation of cluster analysis', *Computational and Applied Mathematics*, Vol. 20, No. 1, pp.53–65.

Sneed, H.M. and Sneed, S.H. (2003) 'Creating web services from legacy host programs', *5th International Workshop on web Site Evolution*, pp.59–65.

Tatsubori, M. and Takashi, K. (2006) 'Decomposition and abstraction of web applications for web service extraction and composition', in *IEEE International Conference on web Services*, pp.859–868.

Upadhyaya, B., Khomh, F. and Zou, Y. (2012) 'Extracting RESTful services from web applications', *5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 17–19 December, Taipei, Taiwan.

Upadhyaya, B., Zou, Y., Xiao, H., Ng, J. and Lau, A. (2011) 'Migration of SOAP-based Services to RESTful services', *Proc. International IEEE Symposium on Web Systems Evolution (WSE)*, IEEE Computer Society Press, Williamsburg, VA, USA, 30 September.

Weather Forecast [online] http://www.theweathernetwork.com/weather/caon0349 (accessed 5 November 2014).

Web Application Description Language (WADL) [online] http:ééwww.w3.org/Submission/wadl/ (accessed 5 November 2014).